CS159

Nathan Sprague

April 3, 2017

Solving the Problem of Fixed Length Arrays

DynamicArray.java↗ DynamicArrayGeneric.java↗
DynamicArrayDriver.java↗

ArrayList

Reminder:
Naming convention for Java Collection types: ArrayList

- Array - Coded using arrays "under the hood".
- List - Implements the List interface↗.
  "An ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list."

- ArrayList API↗

Collections

- Collection - a class that stores multiple elements.
  - (Also a Java interface: Collection API↗)
- We will distinguish between:
  - The interface to a collection - how we interact with the collection.
  - The implementation of the collection - how the data is stored "behind the scenes".

- Java Collections Overview↗
- Java Collections Interfaces Overview↗

Java Arrays

- Note that Java Arrays are in a category by themselves:
  - Not quite objects, not quite primitive types.
  - An array is NOT an object of type array
    - Has no methods.
    - cannot be subclassed.
    - *does* have fields: `myArray.length`
- Advantages:
  - efficient.
  - familiar(?) syntax borrowed from other languages.
- Disadvantages:
  - Fixed length.
  - Awkwardly different from all other collections.

Question

```
1 whichCourse["Nathan"] = "CS159"
2 System.out.println(whichCourse["Nathan"]);
```

1. Does not compile.
2. Compiles, but throws an exception at run time.
3. Runs without error.

(Assuming `whichCourse` is properly initialized.)

Question

```
1 whichCourse["Nathan"] = "CS159"
2 System.out.println(whichCourse["Nathan"]);
```

1. Does not compile.
2. Compiles, but throws an exception at run time.
3. Runs without error.

(Assuming `whichCourse` is properly initialized.)

- Too bad. This would be handy.

HashMap

Recall the Naming Convention: HashMap

- Map - Implements the Map interface↗.
  - A Map maps from a "key" object to a "value" object.
  - Also called a Dictionary or Associative Array.
- Hash - Coded using a hash table (Something to look forward to in CS240!)
  - key/value lookups are very efficient
- HashMap API↗

Example:
HashMapDriver.java↗

HashSet

- Set - Implements the Set interface↗.
  - Stores an unordered collection of items.
  - No duplicates allowed.
  - Checks for containment are very efficient.
- HashSet API↗

Using Collections Effectively

Let's say we want a method that returns an ArrayList with all duplicate elements removed. This seems reasonable:

```
1  public static ArrayList<String> noDuplicates(ArrayList<String> items)
2  {
3      ArrayList<String> result = new ArrayList<String>();
4      for (String item : items)
5      {
6          if (!result.contains(item))
7          {
8              result.add(item);
9          }
10     }
11     return result;
12 }
```

Issues?

Using Collections Effectively

More code than necessary, not very efficient.

```
1  public static ArrayList<String> noDuplicates(ArrayList<String> items)
2  {
3      ArrayList<String> result = new ArrayList<String>();
4      for (String item : items)
5      {
6          if (!result.contains(item)) // <-- This is slow!
7          {
8              result.add(item);
9          }
10     }
11     return result;
12 }
```

Using Collections Effectively

This is better:

```
1  public static ArrayList<String> noDuplicates(ArrayList<String> items)
2  {
3     return new ArrayList<String>(new HashSet<String>(items));
4  }
```

Or, if you prefer:

```
1  public static ArrayList<String> noDuplicates(ArrayList<String> items)
2  {
3     HashSet<String> setVersion = new HashSet<>(items);
4     ArrayList<String> noDups = new ArrayList<>(setVersion);
5     return noDups;
6  }
```

Warning: order of the items will not be retained.

Iterators

- Iterators provide a common mechanism for iterating through Java Collections.
- An iterator is an object that implements the Iterator Interface↗.

Example:
IteratorDemo.java↗

Iterable

- All classes that implement Collection implement the Iterable interface↗.
- This is the magic sauce behind for-each loops.

```
1 for (String s : someCollection)
2     System.out.println(s);
```

Is (pretty much) just a shorthand for:

```
1 Iterator<String> it = SomeCollection.iterator();
2 String s;
3 while(it.hasNext())
4 {
5   s = it.next();
6   System.out.println(s);
7 }
```

Question

```
1         String[] strings = new String[2];
2         strings[0] = "hello";
3         strings[1] = "bob";
4
5         for (String s : strings)
6             System.out.println(s);
```

1. Does not compile.
2. Compiles, but throws an exception at run time.
3. Runs without error.

Question

```
1  public static void main(String[] args)
2  {
3      String[] strings = new String[2];
4      strings[0] = "hello";
5      strings[1] = "bob";
6      printCollection(strings);
7  }
8
9  public static void printCollection(Iterable collection)
10 {
11     for (Object o : collection)
12     {
13         System.out.println(o);
14     }
15 }
```

1. Does not compile.
2. Compiles, but throws an exception at run time.
3. Runs without error.

Summary: When to Use What

- Need efficient access by position. Know in advance exactly how many elements will be stored.

Summary: When to Use What

- Need efficient access by position. Know in advance exactly how many elements will be stored.
  - Array or ArrayList

Summary: When to Use What

- Need efficient access by position. Know in advance exactly how many elements will be stored.
  - Array or ArrayList
- Need efficient access by position. Don't know how many elements will be stored.

Summary: When to Use What

- Need efficient access by position. Know in advance exactly how many elements will be stored.
  - Array or ArrayList
- Need efficient access by position. Don't know how many elements will be stored.
  - ArrayList

Summary: When to Use What

- Need efficient access by position. Know in advance exactly how many elements will be stored.
  - Array or ArrayList
- Need efficient access by position. Don't know how many elements will be stored.
  - ArrayList
- Need to prevent repeats and efficiently check containment.

Summary: When to Use What

- Need efficient access by position. Know in advance exactly how many elements will be stored.
  - Array or ArrayList
- Need efficient access by position. Don't know how many elements will be stored.
  - ArrayList
- Need to prevent repeats and efficiently check containment.
  - HashSet
  - TreeSet (if ordered iteration is important)

Summary: When to Use What

- Need efficient access by position. Know in advance exactly how many elements will be stored.
  - Array or ArrayList
- Need efficient access by position. Don't know how many elements will be stored.
  - ArrayList
- Need to prevent repeats and efficiently check containment.
  - HashSet
  - TreeSet (if ordered iteration is important)
- Need efficient lookup based on a key.

Summary: When to Use What

- Need efficient access by position. Know in advance exactly how many elements will be stored.
  - Array or ArrayList
- Need efficient access by position. Don't know how many elements will be stored.
  - ArrayList
- Need to prevent repeats and efficiently check containment.
  - HashSet
  - TreeSet (if ordered iteration is important)
- Need efficient lookup based on a key.
  - HashMap
  - TreeMap (if ordered iteration by key is important)